Query 1# This code displays the top 3 products that were purchased along with a particular product in the past orders. It also displays the number of times the second product was ordered along with the first product. This could help the customers locate related products easily, and help the business sell more products in a single order.

```
WITH
order_products AS(
    SELECT orderid, p.productname
    FROM  items_in_orders io
    JOIN items i ON io.serialnumber=i.serialnumber
    JOIN products p ON i.productid=p.productid
),
ordered_together AS(
    SELECT a.productname, b.productname AS second_product, COUNT(*) AS
times_ordered_together,
    row_number() OVER(PARTITION BY a.productname ORDER BY COUNT(*) DESC) as rn
    FROM order_products a
    JOIN order_products b ON a.orderid=b.orderid
    WHERE a.productname<>b.productname
    GROUP BY a.productname, b.productname
)
SELECT productname, second_product, times_ordered_together, rn AS
rank_of_second_product
FROM ordered_together
WHERE rn<=3;
```

Query 2# This query find the top deals on best selling products. Best selling products are described as the products that are sold more times than the average number of times a product is sold. This helps customers find the best deals.

```
--2. Top deals on best selling products
WITH
average_times_ordered AS(
    SELECT ROUND(AVG(TotalNumberOfTimesOrdered),2) AS
AverageNumberOfTimesOrdered
    FROM (
    SELECT p.productid, COUNT(*)AS TotalNumberOfTimesOrdered
    FROM items_in_orders io
    JOIN items i ON io.serialnumber = i.serialnumber
    JOIN products p ON p.productid = i.productid
    GROUP BY p.productid)t
    ),
best_selling_products AS(
    SELECT p.productid, p.productname, p.priceperunit, COUNT(*) AS
NumberOfTimesOrdered
    FROM items_in_orders io
    JOIN items i ON io.serialnumber = i.serialnumber
    JOIN products p ON p.productid = i.productid
    HAVING COUNT(*) >  (SELECT AverageNumberOfTimesOrdered FROM
average_times_ordered)
```

```
    GROUP BY p.productid, p.productname, p.priceperunit
    )
SELECT * FROM (
    SELECT bsp.productname, d.discountPercentage,
(bsp.priceperunit*d.discountPercentage)/100 AS discountValue,
    row_number() OVER(ORDER BY (bsp.priceperunit*d.discountPercentage)/100
DESC) AS rank_of_values
    FROM deals_on_products dop
    JOIN best_selling_products bsp ON dop.productid = bsp.productid
    JOIN deals d ON dop.dealid = d.dealid
    )
WHERE rank_of_values<=3;
```

Query 3# The code displays if a product is eligible for free delivery for a particular customer. The product is eligible for free delivery, if there are items available for that product in a store in the same pincode as the customer. This helps customers easily find products that are eligible for free delivery and they are more likely to order those products.

```
SELECT CASE WHEN avail>0 THEN 'Yes' ELSE 'No' END AS "Eligible for free
delivery?" FROM (
    SELECT COUNT(*) AS avail
    FROM customer_address ca
    JOIN stores s ON ca.zipcode=s.zipcode
    JOIN items i ON s.storeid=i.storeid
    JOIN products p ON i.productid=p.productid
    WHERE ca.custid='100007' AND availability=1 AND p.productid='1006'
) ;
```

Query 4# The code displays the total money saved by the customer through all deals or coupons used by him in the past orders.This promotes brand loyalty and encourage repeat orders from customers.

```
WITH discount AS(
    SELECT custid, SUM((p.priceperunit*d.discountPercentage)/100) AS
discountValue
    FROM orders o
    JOIN items_in_orders io ON io.orderid = o.orderid
    JOIN items i ON i.serialnumber = io.serialnumber
    JOIN products p ON p.productid = i.productid
    JOIN deals_on_products dop ON dop.productid = p.productid
    JOIN deals d ON dop.dealid = d.dealid
    GROUP BY custid),
coupon AS(
    SELECT custid, SUM(coupondiscount) AS couponDiscount
    FROM orders
    GROUP BY custid)
SELECT discount.custid, discount.discountValue + coupon.couponDiscount AS
TotalMoneySaved
FROM discount JOIN coupon ON discount.custid = coupon.custid;
```

Query 5# The code displays the products for which there are 5 or less items left in stock. This is used to display an alert on the website for products which are low on stock. It helps customers buy a product that they don't wnat to miss.

```
SELECT p.productid, COUNT(*)
FROM products p
JOIN items i ON p.productid=i.productid
WHERE availability=1
GROUP BY p.productid
HAVING COUNT(*)<=5;
```

Query 6# This code is used to display the top products purchased in the current season. It will check the top products sold in the same season in the past years.It helps customers easily find products that are sold in a season. Eg. Gardening related products are sold relatively frequently in summer.

```
WITH display_current_season As (
    SELECT CASE
    WHEN EXTRACT(MONTH FROM current_date) IN (6,7,8) THEN 'Summer'
    WHEN EXTRACT(MONTH FROM current_date) IN (9,10,11) THEN 'Fall'
    WHEN EXTRACT(MONTH FROM current_date) IN (12,1,2) THEN 'Winter'
    WHEN EXTRACT(MONTH FROM current_date) IN (3,4,5) THEN 'Spring' END AS
current_season
    FROM dual
),
order_season AS (
    SELECT o.orderid, productid, CASE
    WHEN EXTRACT(MONTH FROM orderdate) IN (6,7,8) THEN 'Summer'
    WHEN EXTRACT(MONTH FROM orderdate) IN (9,10,11) THEN 'Fall'
    WHEN EXTRACT(MONTH FROM orderdate) IN (12,1,2) THEN 'Winter'
    WHEN EXTRACT(MONTH FROM orderdate) IN (3,4,5) THEN 'Spring' END AS
order_season
    FROM orders o
    JOIN items_in_orders io ON o.orderid=io.orderid
    JOIN items i ON io.serialnumber=i.serialnumber
),
display_season_products AS (
    SELECT current_season, productid, COUNT(DISTINCT orderid) AS total_orders,
    row_number() OVER(PARTITION BY current_season ORDER BY COUNT(DISTINCT
orderid) DESC) AS rank_of_product
    FROM display_current_season dcs
    JOIN order_season os ON dcs.current_season=os.order_season
    GROUP BY current_season, productid
)
SELECT * FROM display_season_products WHERE rank_of_product<=3;
```

Query 7# The code is used to display an average of the ratings given to a product by customers who have purchased it in the past. The average will only be displayed if there are a minimum of 5 reviews available for that product.It helps the customer get an idea about the quality of the product.

```
SELECT rd.productid, ROUND(AVG(ratings),2) AS avg_rating
FROM review_details rd
JOIN orders o ON rd.custid=o.custid
JOIN items_in_orders io ON o.orderid=io.orderid
JOIN items i ON io.serialnumber=i.serialnumber
GROUP BY rd.productid
HAVING COUNT(DISTINCT reviewid)>=5;
```

Query 8# The code displays the average query resolution time for the agent assigned to his query. This can be displayed to the customer for him to know when his query will be resolved. It helps in setting the right expectations to customers in terms of when their query would be resolved.

```
SELECT cc.agentid, agentfname,
COALESCE(ROUND(AVG((resolutiondate-querydate)*24)),0) AS ResolutionTimeinHours
FROM customer_care cc
JOIN agents a ON a.agentid = cc.agentid
WHERE custid='100007' AND cc.agentid='104'
GROUP BY cc.agentid, agentfname;
```

Query 9# The code displays the items ordered by the customer which are still eligible to be returned. Each product has a specific return period (can be 7 days for some products, 14 or 28 for some and so on). We also display the days remaining to request a return for the item. We also check that item has not been returned already.

```
SELECT o.orderid, p.productid, p.productname, i.serialnumber ,
ROUND((current_date - actualdeliverydate),0) as return_period_ending_in
FROM orders o
JOIN items_in_orders io ON o.orderid=io.orderid
JOIN items i ON io.serialnumber=i.serialnumber
JOIN products p ON i.productid=p.productid
WHERE (current_date - actualdeliverydate)<=returnperiod
AND orderstatus ='Delivered' AND custid='100007'
AND NOT EXISTS
(
    SELECT orderid, serialnumber
    FROM returns r
    WHERE o.orderid=r.orderid AND io.serialnumber=r.serialnumber
);
```

Query 10# The code is used to recommend the next best products that can be purchased by a customer based on the products purchased in his last order. The next best products are determined by checking what products were purchased in the next order by other customers in the past.

```
WITH all_orders AS (
    SELECT o.orderid, custid, orderdate, productname
    FROM orders o
    JOIN items_in_orders io ON o.orderid=io.orderid
    JOIN items i ON io.serialnumber=i.serialnumber
    JOIN products p ON i.productid=p.productid
),
next_best_product AS (
    SELECT custid, productname , orderid,
    LEAD(productname) OVER(PARTITION BY custid ORDER BY ORDERDATE) AS "Next
Ordered",
    LEAD(orderid) OVER(PARTITION BY custid ORDER BY ORDERDATE) AS "Next Order
ID"
    FROM all_orders
)
SELECT productname, "Next Ordered" , count(*)
FROM next_best_product
WHERE "Next Ordered" IS NOT NULL AND orderid<>"Next Order ID"
AND productname IN
(
    SELECT productname
    FROM ORDERS o
    JOIN items_in_orders io ON o.orderid=io.orderid
    JOIN items i ON io.serialnumber=i.serialnumber
    JOIN products p ON i.productid=p.productid
    WHERE custid='100018' AND
    orderdate =
        (SELECT MAX(orderdate) FROM orders where custid='100018')
)
GROUP BY productname, "Next Ordered"
ORDER BY COUNT(*) DESC
FETCH FIRST 3 ROWS ONLY
;
```